分治算法时间性能分析

分治算法回顾

基本思想

- 把一个规模大的问题划分为规模较小的子问题，然后分而治之，最后通过合并子问题的解来得到原问题的解。

基本步骤

- 分解原问题：将原问题划分为一些相互独立的子问题；
- 求解子问题：递归地求解每个子问题；
- 合并子问题的解：将子问题的解组合成原问题的解。

注意：子问题的独立性是保证分治法效率的关键，如果子问题是不独立的，则要做许多不必要的工作来重复地求解公共子问题。

分治算法时间性能分析

- 设$T(n)$是分治算法在问题规模为n时的运行时间，若问题规模足够小，如$n \le C$（常数），则直接求解的时间为$\Theta(1)$。
- 否则：
  - 设完成子问题划分的时间为$D(n)$
  - 设分解时，原问题被划分为了a个子问题，每个子问题规模的为原问题的$1/b$，则解各子问题的时间为$aT(n/b)$
  - 设合并子问题时间为$C(n)$

$$\therefore T(n)=\begin{cases}\Theta(1) & \text{if } n \le c \ \text{//边界} \\ aT(n/b)+D(n)+C(n) & \text{otherwise // } n/b<n，\text{否则无限递归}\end{cases}$$

  - 例如在归并排序中，$a=b=2$，$D(n)=\Theta(1)$，$C(n)=\Theta(1)$，求解可得$T(n)=\Theta(nlgn)$

递归式与分治法

- 人们从大量实践中发现，在用分治法设计算法时，最好使子问题的规模大致相同。即将一个子问题分成大小相等的k个子问题的处理方法是行之有效的。这种使子问题规模大致相等的做法是出自一种平衡子问题的思想，它几乎总是比子问题规模不等的做法要好。

- 一般地，解递归式时可忽略细节
  - 假定函数参数为整数，如$2T(n/2)$应为$T(\lceil n/2 \rceil)$或者$T(\lfloor n/2 \rfloor)$
  - 边界条件可忽略，当n较小时$T(n) = \Theta(1)$

  这些细节一般只影响常数因子的大小，不会改变量级。所以在求解时，先忽略细节，然后再决定其是否重要。

示例：二分查找算法

基本步骤

- 分解原问题：将有序序列分成两半。
- 求解子问题：与序列的中位数进行比较，如果相等，则直接返回中位数的索引；如果小于中位数，则在中位数前面的子序列中继续查找；否则，在中位数后面的子序列中进行查找。
- 合并子问题的解：返回查找的结果。

分析

- 二分查找问题的规模缩小到一定程度即可容易地解决；
- 该问题可以分解为若干个规模较小的相同问题，且子问题的解可以合并为原问题的解；
- 分解出的各个子问题是相互独立的（即在有序序列的中位数前面或后面查找是独立的子问题）。

示例：二分查找算法（续）

伪代码

ALGORITHM binarySearch(A[0 … n − 1], x)

$left \leftarrow 0$;
$right \leftarrow n - 1$;
while $(left \leq right)$ do
    mid $\leftarrow (left + right)/2$;
    if(x == A[mid]) return mid ;
    if(x > A[mid]) $left \leftarrow mid + 1$;
    else $right \leftarrow mid - 1$;
return -1;

时间复杂度

每执行一次while循环，待搜索数组的大小就会减少一半。因此，在最坏情况下，while循环被执行了$\log n$次。循环体内运算需要$O(1)$时间，因此整个算法在最坏情况下的计算时间复杂性为$O(\log n)$。